第3章 程序设计初步

3.1 堆栈的作用

3.2 算术逻辑运算指令

3.3 分支程序设计

3.4 循环程序设计

3.5 子程序设计

3.1 堆栈的作用

3.1.1 过程调用和返回指令

3.1.2 参数传递

3.1.3 局部变量

汇编语言中的堆栈
就是高级语言中的栈

✓ 堆栈有如下所列的主要用途：

（1）保护寄存器内容或者保护现场；

（2）保存返回地址；

（3）传递参数；

（4）安排局部变量或者临时变量。

3.1.1 过程调用和返回指令

➢过程的概念

✓在汇编语言中，常把子程序称为**过程**（procedure）。C语言中的函数是子程序，也就是汇编语言中的过程。

✓调用子程序（过程、函数）在本质上是控制转移，它与无条件转移的区别是调用子程序要考虑返回。

✓处理器提供专门的过程调用指令和过程返回指令。通常，过程调用指令用于由主程序转移到子程序，过程返回指令用于由子程序返回到主程序。

3.1.1 过程调用和返回指令

➢演示程序dp31

主程序和子程序
调用和返回

```
#include <stdio.h>
_fastcall  int  cf211(int x, int y)
{
    return  (2 * x + 5 * y + 100);
}
//
int  main( )
{
    int  val;
    val = cf211(23, 456);
    printf("val=%d\n", val);
    return  0;
}
```

由寄存器传递参数

3.1.1 过程调用和返回指令

➢示例分析

函数cf211目标代码

2 * x + 5 * y + 100

ECX传递参数x

EDX传递参数y

```
cf211    PROC                          ;过程开始
    lea    eax, DWORD PTR [edx+edx*4+100]     ;EAX=5*y+100
    lea    eax, DWORD PTR [eax+ecx*2]         ;EAX=EAX+2*x
    ret                                ;返回（返回值在EAX中）
cf211    ENDP                          ;过程结束
```

PROC 和 ENDP

表示过程代码的开始和结束

属于汇编指示（指令）

3.1.1 过程调用和返回指令

➢示例分析

```
_main    PROC                ;过程开始
                             ;val = cf211(23, 456)
    mov    edx, 456          ;//由寄存器EDX传参数y
    mov    ecx, 23           ;//由寄存器ECX传参数x
    call   cf211             ;//调用函数cf211
                             ;printf("val=%d\n", val)
    push   eax               ;把val值压入堆栈
    push   OFFSET   FMTS     ;把输出格式字符串首地址压入堆栈
    call   _printf           ;//调用库函数_printf
    add    esp, 8            ;//平衡堆栈
    ;
    xor    eax, eax          ;由eax传递返回值
    ret                      ;//返回
_main    ENDP                ;过程结束
```

OFFSET运算符
返回偏移值

3.1.1 过程调用和返回指令

➢过程调用指令（CALL）

段内　直接　调用指令

✓ 过程调用指令的一般格式

CALL　LAB

LAB可以是程序中的一个标号，也可以是一个过程名。

段内直接调用指令进行如下具体操作：

（1）把返回地址偏移（EIP内容）压入堆栈

（2）使得EIP之内容为目标地址偏移，从而实现转移

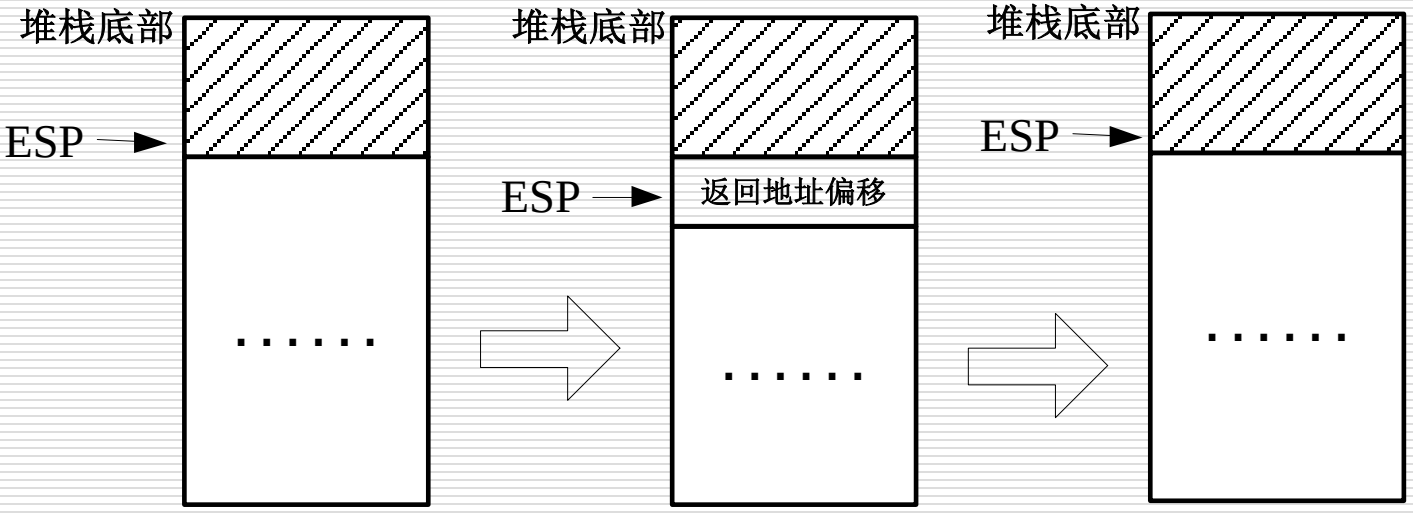
第二步与无条件转移指令的操作相同。与无条件转移指令相比，过程调用指令CALL只是多了第一步（保存返回地址）。

返回地址：紧随过程调用指令的下一条指令的地址（有效地址）
目标地址：子程序开始处的地址（有效地址）

3.1.1 过程调用和返回指令

➢过程调用指令

执行 段内调用指令
堆栈变化示意



(a)调用前的堆栈　(b)调用后的堆栈　(c)返回后的堆栈

3.1.1 过程调用和返回指令

➢过程返回指令（RET）

段内 返回指令

✓ 过程返回指令的一般格式

```
RET
```

该指令从堆栈弹出地址偏移，送到指令指针寄存器EIP。

- 过程返回指令用于从子程序返回到主程序。
- 在执行该指令时，从堆栈顶弹出返回地址，并转移到所弹出的地址，这样就实现了返回。
- 通常，这个返回地址就是在执行对应的调用指令时所压入堆栈的返回地址。
- 过程返回指令的使用应该与过程调用指令所对应。

3.1.1 过程调用和返回指令

➢演示程序dp32

演示调用子程序

```
#include  <stdio.h>
char  string[] = "abcde";
int  main()
{    _asm  {  //嵌入汇编代码
        LEA   ESI, str
        MOV   AX, [ESI]
        CALL  TUPPER              //调用子程序tupper
        MOV   [ESI], AX
        MOV   AX, [ESI+2]
        CALL  TUPPER              //调用子程序tupper
        MOV   [ESI+2], AX
        MOV   AL, [ESI+4]
        CALL  UPPER               //调用子程序upper
        MOV   [ESI+4], AL
    }
    printf("%s\n", str);          //显示为ABCDE
    return  0;
```

3.1.1 过程调用和返回指令

- 演示程序dp32

```
__asm {
UPPER:                          子程序入口标号
    CMP   AL, 'a'
    JB    UPPER2
    CMP   AL, 'z'
    JA    UPPER2
    SUB   AL, 20H
UPPER2:
    RET   //返回
//
TUPPER:                         子程序入口标号
    CALL  UPPER                 //调用子程序
    XCHG  AH, AL
    CALL  UPPER                 //调用子程序
    XCHG  AH, AL
    RET   //返回
}
} //main
```

3.1.2 参数传递

➢参数传递

- ✓ 主程序在调用子程序时，往往要向子程序传递一些参数；同样，子程序运行后也经常要把一些结果返回给主程序。主程序与子程序之间的这种信息传递被称为**参数传递**。
- ✓ 把由主程序传给子程序的参数称为子程序的**入口参数**，把由子程序传给主程序的参数称为子程序的**出口参数**。
- ✓ 一般而言，子程序既有入口参数，又有出口参数。但有的子程序只有入口参数，而没有出口参数；少数子程序只有出口参数，而没有入口参数。

3.1.2 参数传递

➢参数传递方法

- ✓ 有多种传递参数的方法：寄存器传递法、堆栈传递法、约定内存单元传递法和**CALL**后续区传递法等。有时可能同时采用多种方法。根据具体情况而事先约定好。
- ✓ **寄存器传递参数**就是把参数放在约定的寄存器中。实现简单和调用方便。只适用于传递参数较少的情形。
- ✓ 堆栈可以用于传递参数。不占用寄存器，也无需额外的存储单元。但较为复杂。

3.1.2 参数传递

➢参数传递方法

- ✓ C语言的函数通常利用堆栈传递入口参数，而利用寄存器传递出口参数。
- ✓ 如果使用堆栈传递入口参数，那么主程序在调用子程序之前，把需要传递的参数依次压入堆栈，然后子程序从堆栈中取入口参数。

3.1.2 参数传递

➢演示程序dp33

演示堆栈传递参数

```
#include  <stdio.h>
int  cf34(int x, int y)
{
    return   (2 * x + 5 * y + 100) ;
}
//
int  main( )
{
    int  val;
    val = cf34(23, 456);
    printf("val=%d\n", val);
    return  0;
}
```

与演示程序**dp31**相同
但没有调用约定 **_fastcall**

3.1.2 参数传递

➢演示程序dp33

函数cf34目标代码

```
cf34    PROC                                    ;过程开始
        push  ebp                               ;把EBP压入堆栈
        mov   ebp, esp                          ;使得EBP指向栈顶
        mov   eax, DWORD PTR [ebp+12]           ;从堆栈取参数y
        mov   ecx, DWORD PTR [ebp+8]            ;从堆栈取参数x
        lea   eax, DWORD PTR [eax+eax*4+100]    ;EAX=5*y+100
        lea   eax, DWORD PTR [eax+ecx*2]        ;EAX=EAX+2*x
        pop   ebp                               ;恢复EBP
        ret                                     ;返回
cf34    ENDP                                    ;过程结束
```

返回值在**EAX**中

3.1.2 参数传递

➢演示程序dp33

演示程序dp33目标代码

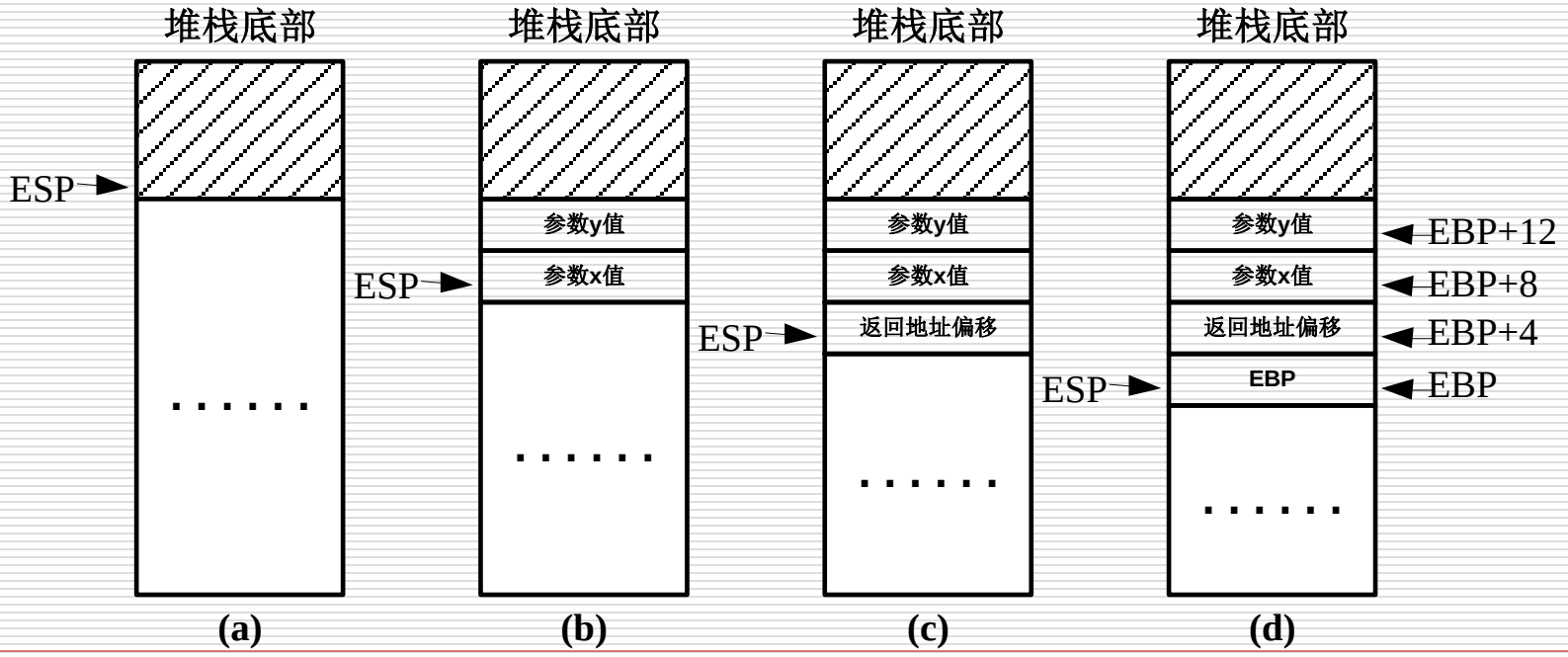
```
_main    PROC              ;过程开始
                           ;val = cf34(23, 456)
    push   456             ;把参数y（000001c8H）压入堆栈
    push   23              ;把参数x（00000017H）压入堆栈
    call   cf34            ;调用函数cf34
                           ;printf("val=%d\n", val)
    push   eax             ;把val值压入堆栈
    push   OFFSET  FMTS    ;把输出格式字符串首地址压入堆栈
    call   _printf         ;调用库函数_printf
    add    esp, 16         ;平衡堆栈
    ;
    xor    eax, eax        ;由eax传递返回值
    ret                    ;返回
_main    ENDP              ;过程结束
```

3.1.2 参数传递

➢堆栈参数分析

堆栈传递参数

堆栈变化示意



3.1.2 参数传递

➢演示函数cf35

观察不同编译选项下的
目标代码

```
int  cf35(int  x, int  y)
{
    if  (x < y)  x = y;
    return  x;
}
```

返回较大值

3.1.2 参数传递

➢演示函数cf35

速度最大化

```
;函数cf35的目标代码
cf35    PROC                    ;表示过程（函数）开始
    push  ebp                   ;
    mov   ebp, esp              ;建立堆栈框架
    mov   eax, DWORD PTR [ebp+8]        ;从堆栈取参数x
    mov   ecx, DWORD PTR [ebp+12]       ;从堆栈取参数y
    cmp   eax, ecx              ;比较x和y（EAX代表x，ECX代表y）
    jge   SHORT ln1cf35         ;如果x大于等于y，就跳转
    mov   eax, ecx              ;实现x=y
ln1cf35:
    pop   ebp                   ;撤销堆栈框架
    ret                         ;返回
cf35    ENDP                    ;表示过程（函数）结束
```

3.1.2 参数传递

➢演示函数cf35

禁用优化

函数cf35的目标代码

```
cf35    PROC                    ;表示过程（函数）开始
    push   ebp
    mov    ebp, esp                              ;建立堆栈框架
    mov    eax, DWORD PTR  [ebp+8]
    cmp    eax, DWORD PTR  [ebp+12]
    jge    SHORT ln1cf35
    mov    ecx, DWORD PTR  [ebp+12]
    mov    DWORD PTR [ebp+8], ecx
ln1cf35:
    mov    eax, DWORD PTR  [ebp+8]
    pop    ebp                                   ;撤销堆栈框架
    ret
cf35    ENDP                    ;表示过程（函数）结束
```

3.1.2 参数传递

➢演示函数cf35

速度最大化
且省略帧指针

```
;函数cf35的目标代码
cf35    PROC                    ;表示过程（函数）开始
    mov   eax, DWORD PTR [esp+4]    ;从堆栈取参数x
    mov   ecx, DWORD PTR [esp+8]    ;从堆栈取参数y
    cmp   eax, ecx                  ;比较之
    jge   SHORT ln1cf35             ;大于等于则跳转
    mov   eax, ecx                  ;使得EAX含较大者
ln1cf35:
    ret
cf35    ENDP                    ;表示过程（函数）结束
```

3.1.3 局部变量

➢局部变量

- ✓ 局部变量是高级语言中的概念。所谓局部变量指对其的访问仅限于某个局部范围。在**C**语言中，局部的范围可能是函数，或者是复合语句。局部变量还有动态和静态之分。
- ✓ **堆栈可以用于安排动态局部变量。**

3.1.3 局部变量

➢演示函数cf36

```
int  cf36(int  x, int  y)
{
    int  z;
    z = x;
    if  (x < y)  z = y;
    return  z;
}
```

返回较大值
刻意安排了局部变量

3.1.3 局部变量

禁用优化，编译所得

➢演示函数cf36

```
cf36      PROC                    ;表示过程（函数）开始
    push  ebp
    mov   ebp, esp                ;建立堆栈框架
                                  ;
    push   ecx                    ;在堆栈中安排局部变量z
                                  ; z = x;
    mov   eax, DWORD PTR  [ebp+8]  ;取得形参x
    mov   DWORD PTR  [ebp-4], eax   ;送到变量z
                                  ; if  (x < y)  z = y;
    mov   ecx, DWORD PTR  [ebp+8]  ;取得形参x
    cmp   ecx, DWORD PTR  [ebp+12]  ;比较x与y
    jge   SHORT  LN1cf36          ;如果x大于y，则跳转
```

3.1.3 局部变量

➢演示函数cf36

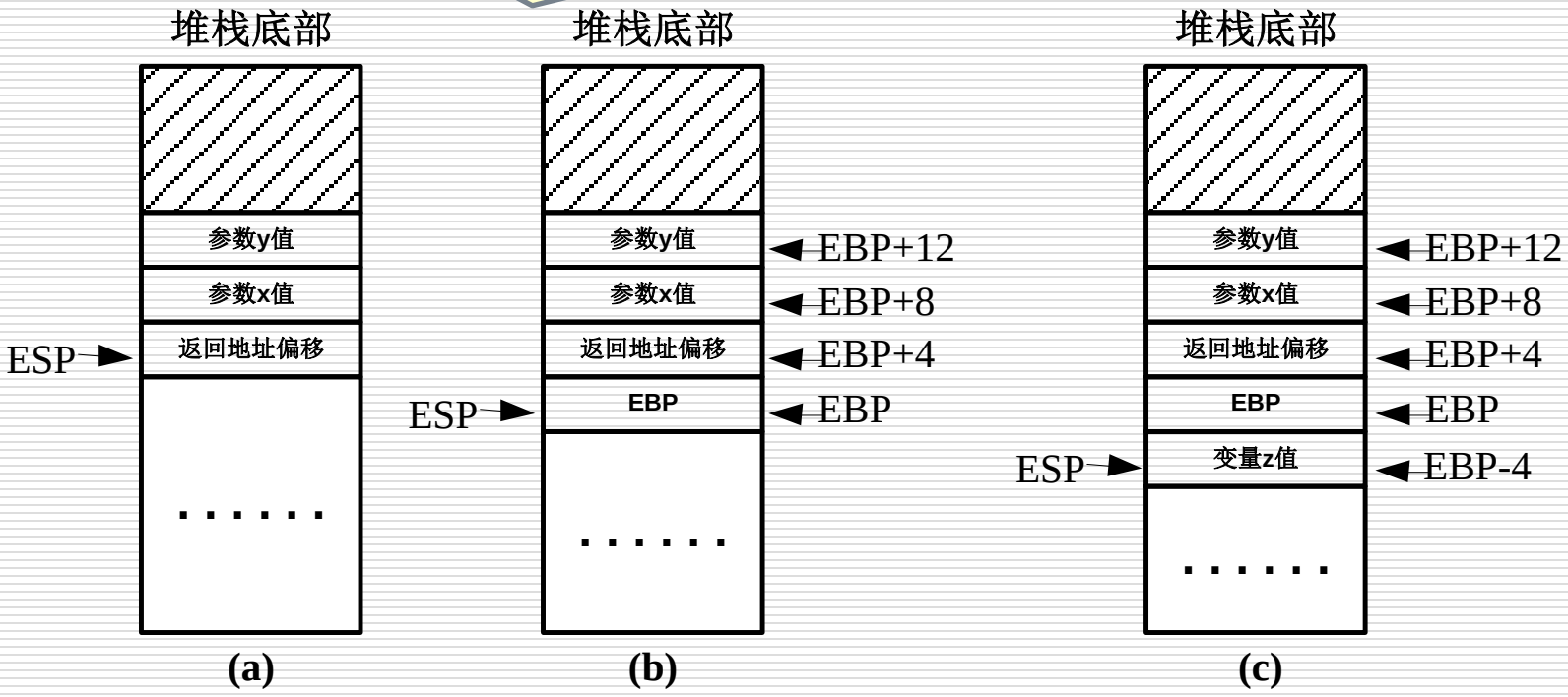
```
cf36     PROC                ;表示过程（函数）开始
    。。。。。。
    mov    edx, DWORD PTR  [ebp+12]    ;取得形参y
    mov    DWORD PTR  [ebp-4], edx     ;送到变量z
LN1cf36:
                                       ; return  z;
    mov    eax, DWORD PTR  [ebp-4]     ;把z送到EAX
                                       ;
    mov    esp, ebp                    ;撤销局部变量z
                                       ;
    pop    ebp                         ;撤销堆栈框架
    ret                                ;返回
cf36     ENDP                ;表示过程（函数）结束
```

3.1.3 局部变量

➢局部变量

堆栈示意

安排局部变量并且由堆栈传递参数



3.1.3 局部变量

➢演示函数cf37

求累加和，
为了演示，安排　2个局部变量

```
int  cf37(int  n)
{
    int  i, sum;
    sum = 0;
    for ( i=1; i <= n; i++ )
        sum += i;
    return  sum;
}
```

3.1.3 局部变量

➢演示函数cf37

函数cf37目标代码

```
cf37    PROC                ;表示过程（函数）开始
        push   ebp
        mov    ebp, esp     ;建立堆栈框
        sub    esp, 8       安排局部变量i和sum
        mov    DWORD PTR [ebp-8], 0        ;sum=0;
        mov    DWORD PTR [ebp-4], 1        ;i=1;
        jmp    SHORT  LN3cf37
```

3.1.3 局部变量

➢演示函数cf37

```
cf37    PROC                ;表示过程（函数）开始
    。。。。。。
LN2cf37:                                ;i++
    mov    eax, DWORD PTR  [ebp-4]      ;取出i
    add    eax, 1
    mov    DWORD PTR  [ebp-4], eax      ;送回i
LN3cf37:                                ;比较i和n
    mov    ecx, DWORD PTR  [ebp-4]
    cmp    ecx, DWORD PTR  [ebp+8]
    jg    SHORT  LN1cf37                ;如果i大于n，则跳转
                                        ;sum += i;
    mov    edx, DWORD PTR  [ebp-8]
    add    edx, DWORD PTR  [ebp-4]
    mov    DWORD PTR  [ebp-8], edx
    jmp    SHORT  LN2cf37
```

3.1.3 局部变量

➢演示函数cf37

```
cf37    PROC                ;表示过程（函数）开始
    。。。。。。
LN1cf37:
    mov    eax, DWORD PTR  [ebp-8]
    mov    esp, ebp
    pop    ebp
    ret
cf37    ENDP
```

准备返回参数

撤销局部变量